An Approach for Scalable Parallel Execution of Ant Algorithms

Franco Cicirelli
DIMES - University of Calabria
Rende (CS), Italy
Email: f.cicirelli@dimes.unical.it

Agostino Forestiero, Andrea Giordano, Carlo Mastroianni
ICAR - CNR
Rende (CS), Italy
Email: {forestiero,giordano,mastroianni}@icar.cnr.it

***Abstract*—This paper presents an approach for the efficient parallel/distributed execution of ant algorithms, based on multi-agent systems. A very popular clustering problem, i.e., the spatially sorting of items belonging to a number of predefined classes, is taken as a use case. The approach consists in partitioning the problem space to a number of parallel nodes. Data consistency and conflict issues, which may arise when multiple agents concurrently access shared data, are transparently handled using a purposely developed notion of logical time. The developer remains in charge only of defining the behavior of the agents modeling the ants, without coping with issues related to parallel/distributed programming and performance optimization. Experimental results show that the approach is scalable and can be adopted to speed up the ant algorithm execution when the problem size is large, as may be in the case of massive data analysis and clustering.**

***Keywords*—Ant algorithms; distributed multi-agent systems; conflict resolution; composed logical time; clustering**

I. Introduction

Bio-inspired algorithms are widely exploited to solve a number of complex problems (combinatorial algorithms, task allocation, routing problems, graph partitioning, etc.) [1] and have been also adopted to provide advanced services in P2P networks [2], Grid systems and Cloud infrastructures. Most biological systems are funded on the *swarm intelligence* paradigm. A number of small and autonomous entities perform very simple operations driven by local information: for example, while searching for food an ant follows a pheromone substance deposited by another ant that has already discovered a food source; a bird adjusts its speed and direction by following the movements of nearby birds. From the combination of such operations a complex and intelligent behavior emerges: ants are able to establish the shortest path towards a food source; birds travel in large flocks and rapidly adapt their movements to the changing characteristics of the environment, etc. [1] [3].

Swarm biological algorithms can be executed by using situated multi-agent systems [4] [5] [6]: the behavior of insects and birds can be reproduced by agents that are situated in a hosting environment (territory) and perform simple operations. Agent-based systems may inherit useful and beneficial properties from biological counterparts, namely: (i) *self-organization*, since decisions are based on local information, i.e., without any central coordinator; (ii) *adaptivity*, since agents can react flexibly to the ever-changing environment; (iii) *stigmergy awareness* [7], since agents are able to interact and cooperate through the modifications of the environment that are induced by their operations.

Parallel/distributed execution of bio-inspired algorithms is often required to cope with the high demand of computational resources needed when the problem becomes more complex or its size increases. In a parallel/distributed scenario, the territory represents a huge shared variable of a concurrent system that needs a careful handling. Territory management requires conflicts and data consistency issues to be addressed. Moreover, frequent access to territory information may easily become a bottleneck impairing the overall system performance and scalability.

In this paper we propose an approach for achieving good performance and scalability in parallel execution of general ant-based algorithms. There are some papers that focus on the same issue [8] [9] [10] [11], but they do not explicitly manage the problem of territory representation and handling. Our approach relies on explicit territory management [12] [13] [14] and a special-purpose notion of *logical time* [12] [15]. Data consistency and conflict management are transparently handled without resorting to lock-based mechanisms and high-level synchronization primitives. The developer of ant-based algorithms remains in charge only of defining the behavior of the agents modeling the ants, without coping with issues related to parallel/distributed programming and performance optimization. Differently from [12], where the purposely developed *logical time* notion was used in the context of event-driven distributed simulation, here the approach is exploited and evaluated for the distributed execution of step-based ant algorithms.

The remainder of the paper is organized as follows. Section II discusses the basic ant algorithm chosen to show the effectiveness of the approach, i.e., the spatial sorting and clustering of items belonging to a number of predefined classes. Then, Section III describes the approach used to parallelize the problem and fasten its execution. The section gives details on the mechanisms used to partition the territory among parallel executing nodes while avoiding conflicts through an ad hoc usage of the logical time concept. Section IV shows the performance of the parallel execution, and reports a speedup analysis when varying the problem size and the number of parallel nodes. Section V describes related work and Section VI concludes the paper.

II. Basic ant algorithm

The objective of the basic ant algorithm presented in [1] is to cluster items in a two-dimensional space. The space is partitioned in cells, forming a two-dimensional grid. Each ant moves hopping between adjacent cells and has visibility over the items deposited in the *visibility area*, which includes the cell where the ant resides and the cells within the *visibility radius* (VR). For example, if the radius is equal to 3, the ant can observe the cells that are at most 3 cells away.

In the simplest scenario, in which items are identical and are spread, the goal is to create regions in which items are accumulated, leaving empty regions in between. This basic version can be specialized in many ways depending on the application. In a very common and significant variation, items belong to a number of predefined different classes, and the objective becomes to spatially sort items, i.e., separate items of different classes and clustering items of the same class. The rest of the paper focuses on such version, which has a large number of applications in several domains, from the organization of physical objects performed by robots to data analysis and clustering in distributed information systems.

Each ant contributes to the reorganization by *picking* and *dropping* items from/to the cells. The ants perform their operations following time-stepped advancements. At each time-step, every ant performs a single operation, either a hop towards an adjacent cell or a drop/pick attempt. The *pick* and *drop* operations are driven by corresponding probability functions, which are inspired by the mechanisms introduced in [16], and later elaborated and discussed in [1] and [17], to emulate the behavior of some species of ants that cluster and sort items in their environment.

The probability of picking an item of a given class from a cell must decrease as items of the same class are accumulated in the visibility area centered in that cell. This ensures that as soon as the equilibrium condition is broken (i.e., items belonging to different classes begin to be accumulated in different areas), a further reorganization of items is increasingly fostered. The P_{pick} probability function, defined in formula (1) below, and inspired by the pick probability defined in [1], aims to achieve the spatial separation of items belonging to different classes.

$$P_{pick} = \left(\frac{k_p}{k_p + f_c} \right)^2 \tag{1}$$

For each class c of items, the fraction f_c is computed as the number of items of class c, accumulated in the cells within the visibility area, divided by the overall number of items of *all* classes that are accumulated in the same area. As the local area accumulates more items of a class, with respect to other classes, f_c increases and the value of the pick probability for this class becomes lower, and vice versa. This has to effect of inducing agents to pick items that are uncommon in the local area, and leave items of the class that is being accumulated. The parameter k_p is assigned a non-negative value and is used to tune the clustering effort. In the tests performed in this work it is set to 0.1, as in [1].

After picking an item, the agent travels the system hopping between adjacent cells, and at any new cell it must decide whether or not to drop the item. Like the pick function, the drop function is first used to break the initial equilibrium and then to strengthen the spatial clustering of items. The *drop* probability for a class, shown in formula (2) below, *increases* as the local area accumulates items of this class. In (2), the fraction f_c is defined as in formula (1), whereas the parameter k_d is set to 0.3 [1].

$$P_{drop} = \left(\frac{f_c}{k_d + f_c} \right)^2 \tag{2}$$

The effectiveness of the clustering algorithm is evaluated through a spatial entropy function, based on the well-known Shannon's formula for the calculation of information content. For each cell l, the local entropy E_l, defined in formula (3), gives an estimation of the extent to which the items have been spatially mapped in the area around l. In (3), f_c is the fraction of items of class c ($c = 1...C$, where C is the number of predefined classes) that are located in the visibility area with respect to the overall number of items located in the same area. The E_l function is normalized so that its value is comprised between 0 and 1. In particular, an entropy value equal to 1 corresponds to the presence of comparable numbers of items of the different classes, whereas a low entropy is obtained when the area centered in l has accumulated a large number of items belonging to one specific class. As shown in formula (4), the overall entropy E is defined as the average of the entropy values E_l computed at all the system cells (the number of cells is equal to N_l).

$$E_l = \frac{\sum_{(c=1...C)} f_c \cdot \lg \frac{1}{f_c}}{\lg C} \tag{3}$$

$$E = \frac{\sum_l E_l}{N_l} \tag{4}$$

The next section shows how this kind of ant algorithm can be transparently executed in a parallel environment while ensuring scalability and preventing consistency issues. It should be remarked here that the approach is generic and can be applied not only to different variants of ant algorithms but also to other types of swarm intelligence algorithms, such as bird flocking, bee colony optimization etc.

III. Parallel/distributed execution of the ant algorithm

As the problem size increases, it may be convenient to parallelize or distribute the execution of ant operations [8] [9] [10] [11]. An ant can be modeled as a *situated agent*, i.e., an agent which owns spatial coordinates and is embedded into the territory (spatial environment) where it moves and lives [4] [6]. The notion of *visibility radius* (VR) introduced in Sec. II is exploited in order to delimit the area within which an ant is able to perceive the surrounding space. The notion of *action radius* (AR) also needs to be introduced. Given an ant located in a specific cell, the AR defines the cell's surrounding area that can be modified by a single operation of the ant. Considering the basic ant operations (i.e. moving from cell to cell and performing pick/drop operations), in the most of ant-based algorithms, the AR may be set to 1.

In a parallel/distributed scenario, the territory is a huge shared variable of a concurrent system. A recurrent access by agents to the territory, can easily become a bottleneck that limits system performance and scalability. In this paper, the whole territory is statically split into equal-sized *regions* as shown in Figure 1. Each region is allocated to a different computing node [13] [18]. An agent is executed by the node associated with the region that includes the cell where the agent is located. Agent migration is required when an ant moves from a region to another.

Partitioning the territory among multiple nodes is a key to avoid that the access to shared content may become a bottleneck. Moreover, splitting the territory favours system scalability in that as the size of the territory increases, more computing nodes can be used to speed up the execution. However, issues relevant to *consistency* and *conflict resolution* on shared data require to be carefully handled.

Usually, conflict resolution and consistency are achieved by resorting to synchronization primitives (e.g. based on locks), which, though, may present two main drawbacks: (i) they may hinder the transparency of the parallelization procedure, since the developer is compelled to cope with the management of such primitives, and (ii) they may negatively impact on performance and scalability. Our approach allows the mentioned issues to be tackled by using a methodology, based on *logical time* [13], which is able to transparently enforce a conflict-free and fair execution order on concurrent actions. This methodology is detailed in Section III-B.

A. Partitioning the territory into regions

When an ant operates on a cell, it needs to retrieve information from a number of neighbor cells, delimited by the visibility radius, and modify the content of the cells delimited

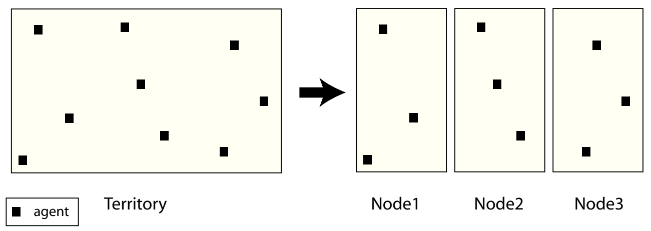


Figure 1. The territory is split into regions that are associated with parallel computing nodes.

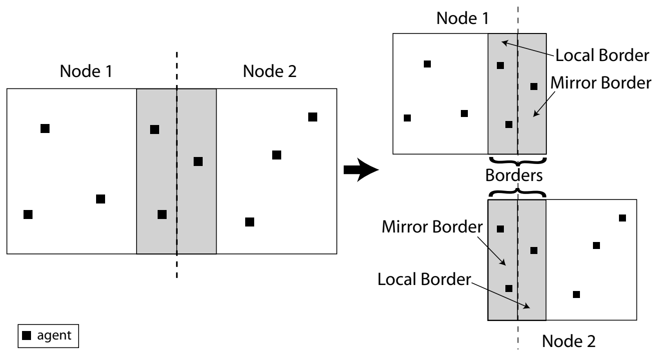
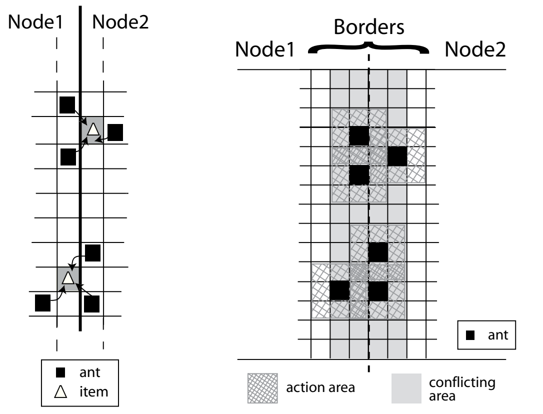


Figure 2. Border areas of two adjacent nodes.

by the action radius. These interactions are *local* when the involved cells are comprised in the same region and hence managed by the same computing node. Conversely, when an ant operation involves some cells belonging to other regions, *remote* interactions – with inter-node information exchange – would be required if this issue were not properly managed. In our approach, in order to avoid remote ant operations, the edge portion of a region is replicated in adjacent nodes. This edge portion is referred to as a *border* of the region, as shown in Figure 2. The border area of a given region (consider Node 1 in the figure) is made up of two distinct parts: the *local border* and the *mirror border*. The first is managed by the local node, and information updates are sent to the mirror border of the adjacent node, i.e., Node 2. Analogously, the mirror border of Node 1 includes information replicated from the local border of Node 2. Agents located in a border area are mirrored by means of *phantom* agents (copy of the original agents that are not actually executed), while items are simply duplicated. The width of borders is determined by the largest between visibility radius and action radius, which in this scenario always corresponds to the visibility radius.

Borders are kept aligned by exchanging update messages between the computing nodes that manage adjacent regions. A single message contains information about all the updates occurred in the local border area during the last time step. This strategy allows all the sensing/acting operations to be performed locally. More details about management of update messages are given in the next section.



(a) A conflicting scenario (b) Potentially conflicting ants and conflicting area

Figure 3. Scenario with ants conflicting on the borders of two nodes

B. A conflict-avoidance mechanism based on logical time

As previously mentioned, splitting the territory and spreading the agents upon different computing nodes raises data consistency issues. This is clarified in Figure 3(a) that shows an example of conflicting scenario where some ants compete to pick up the same items contained in the grey cells. As the ants operate concurrently, two of them may try to pick up the same item: if both pick operations are actually performed, this will lead to an inconsistent state of the algorithm. Each node operates sequentially, i.e., a non-preemptive interleaved execution of ant actions is adopted. Therefore, ants execute concurrently only if they are located on different regions. As shown in Figure 3(b), two ants are *potentially conflicting* when they belong to different regions and their action radii overlap. The *conflicting area* is defined as the portion of the territory that can host potentially conflicting ants (see the grey part of Figure 3(b)).

To prevent conflicts we borrow the notion of *logical time* from the distributed system field. The logical time concept [15] is typically used to prevent causality-constraint violations in distributed systems. In our approach, instead, we exploit it as a tie-breaking mechanism that prevents conflicts [13]. The idea underlying our approach consists in establishing a partial order of ant executions during a given time step such that no potentially conflicting ants will execute concurrently. The problem reduces to assigning, at any given time step, labels (natural numbers) to ants such that potentially conflicting ants are assigned different labels. The labels are used as a logical time that enforces a *conflict-free* execution order.

At each time step, every computing node executes the agents located in the corresponding region, respecting the label ordering discussed here: it executes all the ants with label 1 (the order among them is inessential), then those with label 2, etc. To ensure the algorithm consistency, the nodes must synchronize among them with respect to time steps and ordering labels. In other words, two nodes are not allowed to concurrently execute ant operations related to different time steps or, within a time step, related to different labels. To perform such a synchronization, the update messages exchanged between adjacent nodes, also used to update information on border regions (see Section III-A), are sent at the end of each time step and, within each time step, when completing the executions corresponding to each ordering label. The overhead of the synchronization mechanism can be easily assessed. Let N the number of nodes and b the number of used labels, the number of exchanged messages at each time step is $2 \times N \times b$, since each node sends update messages only to its two adjacent nodes. The number of messages is then linear in the number of exploited computing nodes. This linearity property helps to ensure the scalability of the approach.

An easy fashion to perform a conflict-free labeling of ants is: first assign labels to the cells belonging to the conflicting area and then assign each ant the label of the cell where the ant is located. Figure 4 shows the schema adopted in this work. Every cell is assigned a label between 0 and 7 through the following expression, where x and y are the integer coordinates of the cell:

$$f(x,y) = \left| \left\lfloor \frac{y\%16}{8} \right\rfloor - \left\lfloor \frac{x\%4}{2} \right\rfloor \right| * 4 + \left\lfloor \frac{y\%8}{4} \right\rfloor * 2 + (x+y)\%2$$

It can be noticed that if two cells may host conflicting agents – i.e. two cells belonging to different regions and separated by at most one interposed cell – they are assigned different labels, thus ensuring that the hosting ants will never be executed concurrently.

The choice of adopting the schema of Figure 4, and the generating expression (5), derive, as detailed in [13] [12], from two requirements: (i) limit the number of labels, so as to reduce the number of exchanged messages and the synchronization points and (ii) assign the labels to cells so that two adjacent regions will always execute comparable numbers of operations at each label, i.e., at each logical time. The latter requirement helps to maximize the concurrency degree.

The schema of Figure 4 is established by using a backtracking-based algorithm, which enumerates all the admissible schemes. The adopted one was chosen among those satisfying the above listed requirements. The backtracking algorithm is executed offline with respect to the execution of the ant algorithm so as not to affect system performance. The schema is maintained during the whole algorithm execution but the specific labels are "rotated" at every time step. This avoids to assign static priorities among cells, which would be transferred to the execution order of the hosted ants.

IV. Experimental Results

The presented approach has been evaluated for a typical clustering context, where the ant algorithm is used to sort items belonging to different classes. Our goal is to show that our methodology, while preventing data consistency issues, as

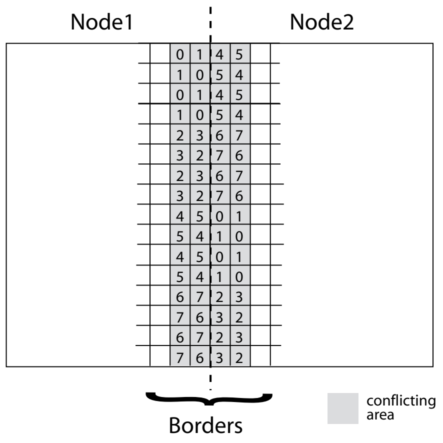


Figure 4. Schema adopted for assigning labels to cells in the conflicting area

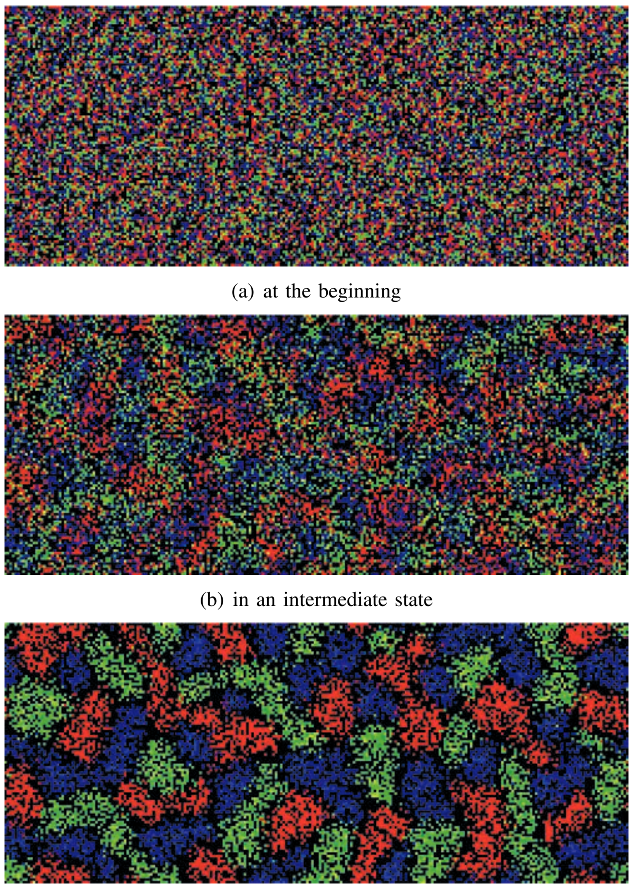


Figure 5. Evolution of the clustering algorithm

explained in the previous section, ensures a high degree of scalability in a wide set of scenarios. The items are spread in a bi-dimensional grid of 120 x 100 cells, and a number of items per cell ranging between 5 and 50. The items belong to a number of classes C between 3 and 9 and are randomly spread over the cells in accordance to a uniform distribution. The number of ant-like agents is proportional to the number of items and spans between 30,000 and 300,000. The scalability is evaluated with two sets of experiments: in the first set we run the algorithm first on a single node and then on three parallel nodes of a cluster, varying the problem size (i.e., the number of items); in the second set, for a given problem size, we varied the number of parallel nodes up to 9. The experiments were carried out on a cluster in which each computing node has CPU Intel(R) Xeon(R) CPU E5-2670 2.60GHz and 128GB RAM. The nodes are interconnected with an Intel Corporation I350 Gigabit Network.

To show how the algorithm clusters the items, Figure 5 reports three snapshots of the system taken before starting the algorithm, in an intermediate state, and when clustering has been achieved. The snapshots are taken for the scenario in which items belong to three classes and the visibility radius VR is set to 10. The three classes correspond to the RGB colors. When a cell contains items of different classes, the color corresponds to the dominant class. The color intensity of a cell is proportional to the number of items of the dominant class.

The value of the overall entropy, as defined in expression (4), is computed every 1000 time steps, and decreases as the algorithm proceeds, confirming its effectiveness. The system is considered stable when 10 successive values of the entropy differ among them no more than 1%: this is used as a stop criteria for the algorithm.

Before analyzing the parallel execution of the algorithm, we analyze its behavior when it is executed on a single node in some of the use cases of interest. Figures 6 and 7 show the behavior when varying, respectively, the number of classes and the visibility radius. Specifically, Figure 6 shows that the entropy decreases from values close to 1 to values lower than 0.2, confirming that the items have been effectively clustered. The curves stop when the algorithm terminates its execution. It is noticed that as the number of classes C increases: (i) the stable value of entropy decreases and (ii) the convergence is slightly faster. In these tests, the visibility radius VR was set to 5. The value of VR can be used to tune the size of the clusters, i.e., of the areas containing items of the same class: larger clusters are obtained with larger values of VR. Figure 7 shows the results of experiments in which the number of classes is set to 3 and the visibility radius ranges between 2 and 10. As the VR value increases, the algorithm needs more time steps to converge, and converges to larger values of entropy.

It should be remarked here that during parallel execution we obtained the same behavior as the one illustrated in Figures 6 and 7. This mirrors the fact that the adopted approach does not impair the algorithm evolution. Differently to what happens elsewhere, e.g. in [11], the behavior of the ant algorithm does not depend on the serialized or parallel execution context.

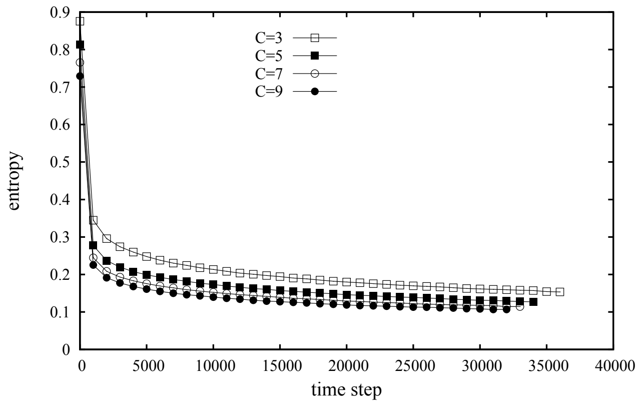


Figure 6. Entropy curves using different number of classes

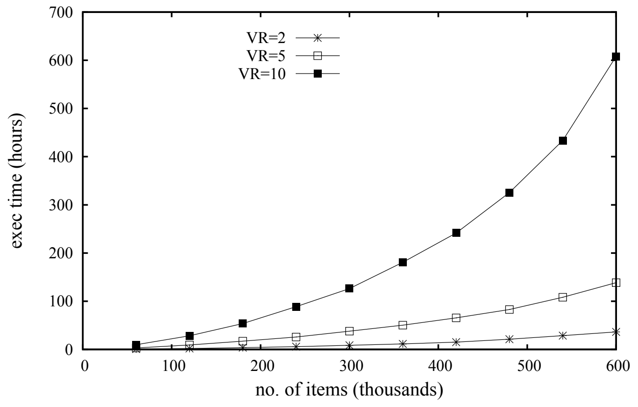


Figure 8. Execution time vs. the number of items

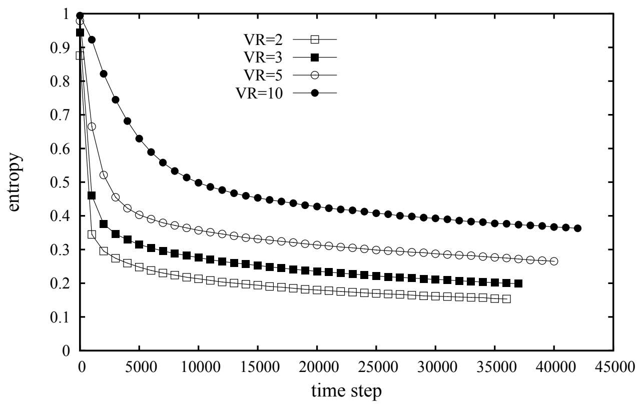


Figure 7. Entropy curves using different visibility radii

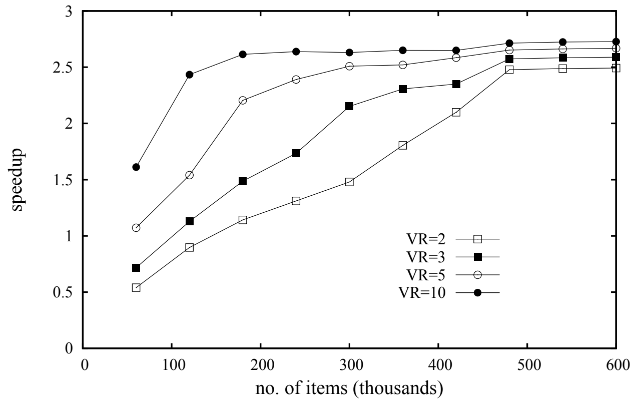


Figure 9. Speedup vs. the number of items

The parallel execution of the algorithm is more and more efficient as the overall computational load increases. In the considered scenario the problem size increases with the number of items and the visibility radius. Indeed, the number of pick/drop attempts is proportional to the number of items while the computational load of a single pick/drop attempt is proportional to the visibility radius, as the radius determines the number of cells involved in the computation of function f_c (see expressions (1) and (2)). It may be remarked here that the computational load does not depend on the number of classes. Figure 8 reports the overall execution time of the algorithm and confirms the mentioned relationships between the number of items, the visibility radius and the computational load.

Performance of parallel execution is assessed by measuring the speedup value, computed as the ratio of the execution time experienced on a single node and the execution time on multiple nodes. Figure 9 reports the speedup on three parallel nodes vs. the number of items, when varying the visibility radius. The algorithm scales very well: as the number of items increases up to 600,000, the speedup value increases up to values between 2.5 (with VR=2) and 2.7 (with VR=10). It is also noticed that, as the visibility radius increases, the steady value of speedup is achieved for smaller values of the number of items. Such behavior is consistent with the computational load analysis discussed before.

Finally, we analyzed the speedup when parallelizing the execution on up to 9 nodes. Figure 10 reports the values of speedup vs. the number of computing nodes and for different problem sizes, when setting the visibility radius to 10. Experimental results reported in Figures 9 and 10 confirm the good scalability and performance of the approach since: (i) the speedup increases with the number of nodes; (ii) for a given

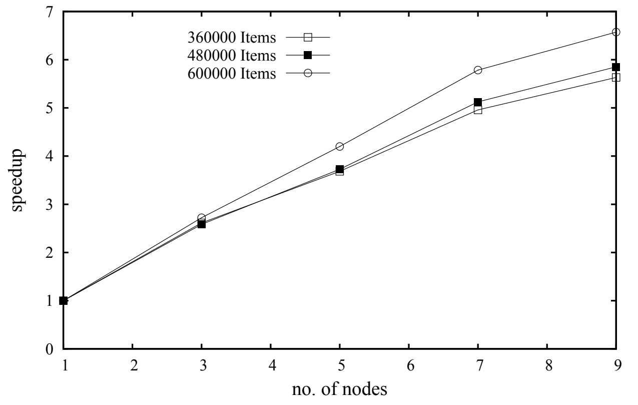


Figure 10. Speedup vs. the number of computing nodes

number of nodes, the speedup increases with the problem size.

V. Related Work

This paper presents an approach that can be used to improve performance and achieve good scalability when a wide class of algorithms are ported and executed in parallel architectures. The approach is applied to ant algorithms, a class of agent systems that aim to solve very complex problems, ranging from the management of distributed systems to routing problems and multi-parameter optimization, by imitating the behavior of some species of ants [1].

Parallel/distributed versions of ant algorithms became popular in the last decade. A recent survey [9] focuses on parallel implementations of Ant Colony Optimization solutions, which aim to improve the efficiency of population-based meta-heuristics. By splitting the population into several processing elements, parallel implementations of meta-heuristics allow to reach high quality results in a reasonable execution time, even when facing hard-to solve optimization problems [19]. In [10], a mathematical model is presented that helps to parallelize a generic Ant Colony Optimization problem while minimizing the workload imbalance among the machines, with the objective of reducing the overall execution time. In [11] an ant algorithm is parallelized using the MapReduce programming model: the input dataset is partitioned into a number of parallel nodes, and partial results are collected by a central controller node. However, with this approach the parallel nodes are isolated, and mobile agents (ants) are not allowed to migrate from one node to another, which limits their movements and hinders the faithful implementation of the ant paradigm.

When the problem size is relevant, it may be useful to exploit computing nodes belonging to heterogeneous systems. In such a case, a tailored framework must be designed to cope with the issues related to a distributed scenario. A configurable distributed architecture was proposed in [20] in order to provide an intuitive and simple mapping of ant colony optimization algorithms in a distributed environment. In this approach, the physical environment of ants is represented and implemented as a distributed multi-agent system, and the movements of ants are modeled through messages that are exchanged asynchronously among the agents. The approach was generalized in [21] to distribute Swarm Intelligence (SI) algorithms that solve graph search problems on a computer network. This work proposes a novel distributed framework, for a class of SI algorithms, which better exploits the inherently distributed nature of these algorithms.

AntNet [8] is a distributed multi-agent routing algorithm proposed for wired datagram networks based on the principle of ant colony optimization. In AntNet, each node maintains a routing table and an additional table containing statistics about the traffic distribution over the network. AntNet uses two sets of homogeneous mobile agents, forward ants and backward ants, to collect information about traffic distribution and update the routing tables. Multi-agent ant algorithms were also used to manage distributed information systems, Grids, and P2P networks. A self-organizing distributed algorithm, *So-Grid*, was proposed to cluster objects according to their classes, so as to facilitate their management and speed up their discovery in a Grid environment [22].

The approaches mentioned so far are focused on the modeling and execution of bio-inspired algorithms but do not explicitly manage the problem of territory representation and handling. Situated multi-agent systems are systems in which the behavior of agents is strongly influenced by their positions in the territory and by their interactions with the surrounding environment [23]. The management of the shared state representing the territory may become a bottleneck, limiting the overall performance when situated agent systems are executed in a parallel/distributed scenario [24] [25].

The concept of *spheres of influence* is introduced in [26] in order to manage shared state in a distributed scenario. The purpose is to favor locality by putting information close to the agent that uses the information during execution. Spheres of influence are dynamically determined on the basis of the mutual interactions among agents and information. In [27] shared data is maintained in a tuple-space. The tuple-space is partitioned by following a hierarchical schema based on the spheres of influence so as to avoid bottleneck in managing the shared data. In [28] the concept of *synchronization regions* is introduced to resolve conflicts among concurrent actions and to reduce synchronization cost in a distributed setting. A *region* is a group of agents that act simultaneously and independently from other agents. Regions are determined by a decentralized synchronization algorithm that is executed when actions are performed.

The described approaches need an explicit effort of the developer for the definition and implementation of explicit high-level synchronization mechanisms and primitives used to manage territory and shared data. Conversely, our approach offers a transparent and efficient way to parallelize the execution of ant algorithms, ensuring good performance while relieving the developer from taking care of the issues related to management of shared state, territory handling and performance optimization.

VI. Conclusions

We presented and evaluated an approach that makes on original use of the logical time concept to automate the parallelization of a wide class of bio-inspired algorithms. The approach is applied to a very popular clustering problem, i.e., the spatially sorting of items belonging to different classes, obtained through probabilistic pick and drop operations performed by mobile agents. The paper describes the mechanisms and the policy used to partition the territory among parallel nodes and manage shared data avoiding conflicts and data inconsistency. This relieves the developer from the burden of explicitly defining and managing high level synchronization primitives to cope with the above mentioned issues. The performance of the clustering algorithm was evaluated through the measurement of spatial entropy. Results, assessed in a parallel environment, show that the approach has good scalability properties and can be adopted to speed up the ant algorithm execution when the problem is complex and/or its size is large.

References

[1] E. Bonabeau, M. Dorigo, and G. Theraulaz, *Swarm intelligence: from natural to artificial systems*. New York, NY, USA: Oxford University Press, 1999.

[2] O. Babaoglu, H. Meling, and A. Montresor, "Anthill: A framework for the development of agent-based peer-to-peer systems," in *Proc. of the 22 nd International Conference on Distributed Computing Systems ICDCS'02*. Washington, DC, USA: IEEE Computer Society, 2002, pp. 15–22.

[3] P. Dasgupta, "Intelligent agent enabled peer-to-peer search using ant-based heuristics," in *Proc. of the International Conference on Artificial Intelligence IC-AI'04*, 2004, pp. 351–357.

[4] M. Wooldridge, *An introduction to multi-agent systems*. John Wiley & Sons, Ltd., 2002.

[5] K. Sycara, "Multiagent systems," *Artificial Intelligence Magazine*, vol. 10, no. 2, pp. 79–93, 1998.

[6] J. Ferber, *Multi-Agent Systems: An Introduction to Distributed Artificial Intelligence*. Addison Wesley Longman, 1999.

[7] P. Grasse', "La reconstruction du nid et les coordinations inter-individuelles chez belicositermes natalensis et cubitermes sp. la thorie de la stigmergie : Essai d'interprtation du comportement des termites constructeurs." *Insectes Sociaux*, no. 6, pp. 41–84, 1959.

[8] G. Di Caro and M. Dorigo, "Antnet: Distributed stigmergetic control for communications networks," *J. Artif. Int. Res.*, vol. 9, no. 1, pp. 317–365, December 1998.

[9] M. Pedemonte, S. Nesmachnow, and H. Cancela, "A survey on parallel ant colony optimization," *Applied Soft Computing*, vol. 11, no. 8, pp. 5181 – 5197, 2011.

[10] T. Keskinturk, M. B. Yildirim, and M. Barut, "An ant colony optimization algorithm for load balancing in parallel machines with sequence-dependent setup times," *Computers & Operations Research*, vol. 39, no. 6, pp. 1225 – 1235, 2012.

[11] Y. Yang, X. Ni, H. Wang, and Y. Zhao, "Parallel implementation of ant-based clustering algorithm based on hadoop," in *Proc. of the 3rd Int. Conference on Advances in Swarm Intelligence - Part I*, ser. ICSI'12. Shenzhen, China: Springer-Verlag, October 2012, pp. 190–197.

[12] F. Cicirelli, A. Giordano, and L. Nigro, "Efficient environment management for distributed simulation of large-scale situated multi-agent systems," *Concurrency and Computation: Practice and Experience*, 2014. [Online]. Available: http://dx.doi.org/10.1002/cpe.3254

[13] ——, "Distributed simulation of situated multi-agent systems," in *Proc. of the IEEE/ACM 15th International Symposium on Distributed Simulation and Real Time Applications*, Washington, DC, USA, 2011, pp. 28–35.

[14] D. Weyns, A. Omicini, and J. Odell, "Environment as a first class abstraction in multiagent systems," *Autonomous Agents and Multi-Agent Systems*, vol. 14, no. 1, pp. 5–30, 2007.

[15] L. Lamport, "Time, clocks and the ordering of events in a distributed system," *Communications of the ACM*, vol. 21, no. 7, pp. 558–565, 1978.

[16] J. L. Deneubourg, S. Goss, N. Franks, A. Sendova-Franks, C. Detrain, and L. Chrétien, "The dynamics of collective sorting robot-like ants and ant-like robots," in *Proc. of the First International Conference on Simulation of Adaptive Behavior on From Animals to Animats*. Cambridge, MA, USA: MIT Press, 1990, pp. 356–363.

[17] M. Martin, B. Chopard, and P. Albuquerque, "Formation of an ant cemetery: swarm intelligence or statistical accident?" *Future Generation Computer Systems*, vol. 18, no. 7, pp. 951–959, 2002.

[18] F. Cicirelli, A. Furfaro, and L. Nigro, "An agent infrastructure over HLA for distributed simulation of reconfigurable systems and its application to UAV coordination," *SIMULATION, Trans. of SCS*, vol. 85, no. 1, pp. 17–32, 2009.

[19] E. Alba, *Parallel metaheuristics: a new class of algorithms*. John Wiley & Sons, 2005, vol. 47.

[20] S. Ilie and C. Badica, "Multi-agent approach to distributed ant colony optimization," *Science of Computer Programming*, vol. 78, no. 6, pp. 762 – 774, 2013.

[21] ——, "Multi-agent distributed framework for swarm intelligence," *Procedia Computer Science*, vol. 18, pp. 611 – 620, 2013, 2013 International Conference on Computational Science. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1877050913003682

[22] A. Forestiero, C. Mastroianni, and G. Spezzano, "So-grid: A self-organizing grid featuring bio-inspired algorithms," *ACM Transactions on Autonomous and Adaptive Systems*, vol. 3, no. 2, May 2008.

[23] S. Bandini, S. Manzoni, and C. Simone, "Dealing with space in multi agent systems: a model for situated MAS," in *Proc. of the 1st Int. Joint Conference on Autonomous Agents and Multiagent Systems*, New York, NY, USA, July 2002, pp. 1183–1190.

[24] B. Logan, "Evaluating agent architectures using simulation," in *Evaluating Architectures for Intelligence: Papers from the 2007 AAAI Workshop*. AAAI Press, 2007, pp. 40–43, Technical Report WS–07–04.

[25] D. Pawlaszczyk and S. Strassburger, "Scalability in distributed simulations of agent-based models," in *Proc. of Winter Simulation Conference (WSC)*, Austin, TX, USA, 2009, pp. 1189–1200.

[26] B. Logan and G. Theodoropoulos, "The distributed simulation of multiagent systems," *Proc. of the IEEE*, vol. 89, no. 2, pp. 174–185, 2001.

[27] M. Lees, B. Logan, R. Minson, T. Oguara, and G. Theodoropoulos, "Modelling environments for distributed simulation," in *Proc. of the 1st International Workshop on Environments for Multi-Agent Systems (E4MAS))*, ser. LNAI, vol. 3374. Springer, 2005, pp. 150–167.

[28] D. Weyns and T. Holvoet, "A formal model for situated multi-agent systems," *Formal Approaches for Multi-Agent Systems, Special Issue of Fundamenta Informaticae*, vol. 63, no. 2, 2004.